消息机制

2020年8月17日 17:18

-
- 消息机制

1. Message Queue/Message Oriented Middleware

		同步	异步
面向消息中间件 (MOM)	IBM MQSeries, Microsoft MSMQ, BEA MessageQ, JBossMQ		X
数据连接	ODBC, JDBC, etc.	X	
远程过程调用 (RPC)	Dubbo,Thrift	X	
对象请求代理 (ORB)	符合CORBA标准的，如Orbix, Visibroker, BEA Objectbroke，Java IIOP；还有Java RMI	X	
交易流程控制 (TPM)	Microsoft Transaction Server (MTS), IBM CICS, IBM Encina, BEA Tuxedo	X	

   a. 对非实时异步需求，引入了消息队列机制
   b. 在应用解耦、流量削锋、日志处理和消息通讯等需求下也可使用
   c. MOM中间件允许一个应用向另一个应用发送消息，而无论该应用是否在线
2. 同步vs异步
   a. 同步优点：易编程、输出立即可知、更易恢复错误 (通常)、更好实时响应 (通常)
   b. 同步缺点：服务必须启动且在线、请求方阻碍占用资源、通常要求面向连接的通信协议
   c. 异步优点：请求无需指定服务器、服务无需在线、由于没有占用, 资源可以释放、可以使用非连接协议
   d. 异步缺点：响应时间不可预测、错误处理通常复杂、难以设计程序
3. 消息机制特点
   a. 消息机制优点：集中精力做应用本身、不用管环境的细节、应用可移植可扩展
   b. 特性：应用程序可以运行在不同时间、应用结构或数量没有限制、屏蔽底层环境（操作系统、编程语言、通信协议）差异
4. 消息概念
   a. 消息：信息的载体
   b. 消息协议：描述消息的统一的格式称之为消息协议。
      i. XMPP：基于XML，用于IM系统的开发
      ii. Stomp：结构简单
      iii. AMQP：经典
   c. 消息队列：消息从某一端发出后，首先进入一个容器进行临时存储，当达到某种条

件后，再由这个容器发送给另一端。这个容器的一种具体实现就是消息队列

5. Java Message Service（JMS）：SUN提出的旨在统一各种MOM（Message-Oriented Middleware ）系统接口的规范，它包含点对点（Point to Point，PTP）和发布/订阅（Publish/Subscribe，pub/sub）两种消息模型，提供可靠消息传输、事务和消息过滤等机制
    a. 和MQ的关系
        i. JMS不是消息队列，更不是某种消息队列协议
        ii. 是一套规范的JAVA API 接口
        iii. JMS和消息中间件厂商无关，需要各个厂商进行实现，大部分消息中间件产品都支持JMS 接口规范
        iv. 可以使用JMS API来连接Stomp协议的产品（例如ActiveMQ）
        v. 类似与JDBC和mysql的关系
    b. 点对点模型(基于队列)
        i. 每个消息只能有**一个消费者。**消息的生产者和消费者之间没有时间上的相关性：可以有多个发送者，但只能被一个消费者消费
        ii. 一个消息只能被一个接受者接受一次
        iii. 生产者把消息发送到队列中(Queue)，这个队列可以理解为电视机频道(channel)
        iv. 在这个消息中间件上有多个这样的channel
        v. 接受者无需订阅，当接受者未接受到消息时就会处于阻塞状态
    c. 发布者/订阅者模型（基于主题的）
        i. 每个消息可以有多个消费者
        ii. 生产者和消费者之间有时间上的相关性。订阅一个主题的消费者只能消费自它订阅之后发布的消息.
        iii. 允许多个接受者，类似于广播的方式
        iv. 生产者将消息发送到主题上(Topic)
        v. 接受者**必须先订阅**
        vi. 注:持久化订阅者：特殊的消费者，告诉主题，我一直订阅着，即使网络断开，消息服务器也记住所有持久化订阅者，如果有新消息，也会知道必定有人回来消费
    d. JMS体系架构
        i. 提供者：连接面向消息中间件的，JMS接口的一个实现。
        ii. 客户：生产或消费基于消息的Java的应用程序或对象。
            1）生产者：创建并发送消息的JMS客户。
            2）消费者：接收消息的JMS客户。
        iii. 消息：包括可以在JMS客户之间传递的数据的对象。
        iv. 队列：一个容纳那些被发送的等待阅读的消息的区域。
        v. 主题：一种支持发送消息给多个订阅者的机制。
    e. JMS对象模型
        i. 连接工厂（Connection Factory）：客户创建连接的对象

ii. 连接（Connection）：封装好的虚拟连接
iii. 会话（Session）：事务性的上下文
iv. 目的（Destination：Queue、Topic）：指定生产者和消费者的对象
v. 生产者（Message Producer）：Session创建的对象
vi. 消费者（Message Consumer）：Session创建的对象

f. JMS消息存储方式
i. NON_PERSISTENT：禁止固化消息，仅将消息放到内存中，适合消息量较少、可靠性要求不高的系统。
ii. PERSISTENT：固化消息，将消息以文件或者数据库的方式进行固化，可有效提高可靠性，但会降低性能。

g. JMS消息格式
i. StreamMessage：Java原始值的数据流
ii. MapMessage：键-值对
iii. TextMessage：字符串对象（最常用）
iv. ObjectMessage：序列化的 Java对象
v. BytesMessage：字节流

h. JMS消息实体
i. 消息头（必须）：包含用于识别和为消息寻找路由的操作设置。
ii. 消息属性（可选）：包含额外的属性，支持其他提供者和用户的兼容。
iii. 消息体（可选）：消息具体内容。

6. JMS提供者
a. ActiveMQ：apache出品，开源，国际上比较流行，不支持超大规模
i. 最流行的，能力强劲的开源消息总线，是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现
ii. 支持多语言和多协议
iii. 完全支持持久化、事务
iv. 支持内嵌至Spring
v. 支持部署至JBoss、WebLogic等服务器上
vi. 支持集群
vii. 友好管理界面，测试方便

b. RabbitMQ：mozilla出品，开源，不支持事务，erlang语言开发。他同时也实现了应用层AMQP协议
c. RocketMQ：阿里巴巴出品，部分开源，该公司内部大量使用，集群50台，日消息量百亿级
d. SonicMQ：progress出品，商业，部分国内网站使用
e. ZeroMQ：C语言开发，效率极高，但不成熟，开源
f. kafka：擅长大数据\海量日志处理
g. 其他IBM、微软等商业产品

7. ActiveMQ使用
a. 默认端口61616，通过netstat 命令查看该端口是否打开，判断是否正确运行

b. demo
   i. ConnectionFactory创建Connection
   ii. Connection创建JMS Session
   iii. Session创Queue或Topic
   iv. Session创MessageProducer或MessageConsumer

c. 点对点生产者

```
//1.创建连接工厂
ConnectionFactory connectionFactory=new
ActiveMQConnectionFactory("tcp://192.168.25.135:61616");
//2.获取连接
Connection connection = connectionFactory.createConnection();
//3.启动连接
connection.start();
//4.获取session  (参数1：是否启动事务,参数2：消息确认模式)
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
//5.创建队列对象
Queue queue = session.createQueue("test-queue");
//6.创建消息生产者
MessageProducer producer = session.createProducer(queue);
//7.创建消息
TextMessage textMessage = session.createTextMessage("欢迎");
//8.发送消息
producer.send(textMessage);
//9.关闭资源
producer.close();
session.close();
connection.close();
```

d. 点对点消费者

```
//1.创建连接工厂
ConnectionFactory connectionFactory=new
ActiveMQConnectionFactory("tcp://192.168.25.135:61616");
//2.获取连接
Connection connection = connectionFactory.createConnection();
//3.启动连接
connection.start();
//4.获取session  (参数1：是否启动事务,参数2：消息确认模式)
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
//5.创建队列对象
Queue queue = session.createQueue("test-queue");
//6.创建消息消费
MessageConsumer consumer = session.createConsumer(queue);
//7.监听消息
consumer.setMessageListener( message -> {  // MesseageListener
      TextMessage textMessage = (TextMessage)message;
      try {
            double pay = Double.parseDouble(textMessage.getText());
            System.out.println("尝试扣款："+pay);
            if (pay < accountBalance) {
                  accountBalance -= pay;
                  send(true);
            } else
                  send(false);
      } catch (JMSException e) {
            e.printStackTrace();
      }});
//8.等待键盘输入
System.in.read();
//9.关闭资源
consumer.close();
session.close();
connection.close();
```

e. 发布订阅生产者

```
//1.创建连接工厂
ConnectionFactory connectionFactory=new
ActiveMQConnectionFactory("tcp://192.168.25.135:61616");
//2.获取连接
```

```
Connection connection = connectionFactory.createConnection();
//3.启动连接
connection.start();
//4.获取session  (参数1：是否启动事务,参数2：消息确认模式)
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
//5.创建主题对象
Topic topic = session.createTopic("test-topic");
//6.创建消息生产者
MessageProducer producer = session.createProducer(topic);
//7.创建消息
TextMessage textMessage = session.createTextMessage("欢迎");
//8.发送消息
producer.send(textMessage);
//9.关闭资源
producer.close();
session.close();
connection.close();
```

f. 发布订阅消费者

```
//1.创建连接工厂
ConnectionFactory connectionFactory=new
ActiveMQConnectionFactory("tcp://192.168.25.135:61616");
//2.获取连接
Connection connection = connectionFactory.createConnection();
//3.启动连接
connection.start();
//4.获取session  (参数1：是否启动事务,参数2：消息确认模式)
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
//5.创建主题对象
//Queue queue = session.createQueue("test-queue");
Topic topic = session.createTopic("test-topic");
//6.创建消息消费
MessageConsumer consumer = session.createConsumer(topic);
//7.监听消息
consumer.setMessageListener(new MessageListener() {
      public void onMessage(Message message) {
      TextMessage textMessage=(TextMessage)message;
      try {
            System.out.println("接收到消息："+textMessage.getText());
      } catch (JMSException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
      }}});
//8.等待键盘输入
System.in.read();
//9.关闭资源
consumer.close();
session.close();
connection.close();
```